SYMBOLIC EXECUTION

Key idea

Reason about behavior of program by”executing” it with symbolic values

Originally proposed by James King (1976, CACM) and Lori Clarke (1976, IEEE TSE)

Practical around 2005 because of advances in constraint solving (SMT solvers)

An example

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

Concrete execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

INITIAL STATE

a	b	c	x	y	z
1	1	1	0	0	0

Concrete execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

Condition: if (a) is true
since a = 1 (truthy in JavaScript)

INITIAL STATE

a	b	c	x	y	z
1	1	1	0	0	0

Concrete execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

INITIAL STATE

a	b	c	x	y	z
1	1	1	0	0	0

Condition: if (a) is true

a	b	c	x	y	z
1	1	1	-2	0	0

Concrete execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

INITIAL STATE

a	b	c	x	y	z
1	1	1	0	0	0

Condition: if (a) is true

a	b	c	x	y	z
1	1	1	-2	0	0

Condition: b > 5 is **false**

Concrete execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

INITIAL STATE

a	b	c	x	y	z
1	1	1	0	0	0

Condition: if (a) is true

a	b	c	x	y	z
1	1	1	-2	0	0

Condition: b > 5 is **false**

a	b	c	x	y	z
1	1	1	-2	0	0

The assertion check

$x + y + z = -2 + 0 + 0 = -2$

$-2 \mathrel{!=} 3$ is true

Symbolic execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

INITIAL STATE

a	b	c	x	y	z
a_0	b_0	c_0	0	0	0

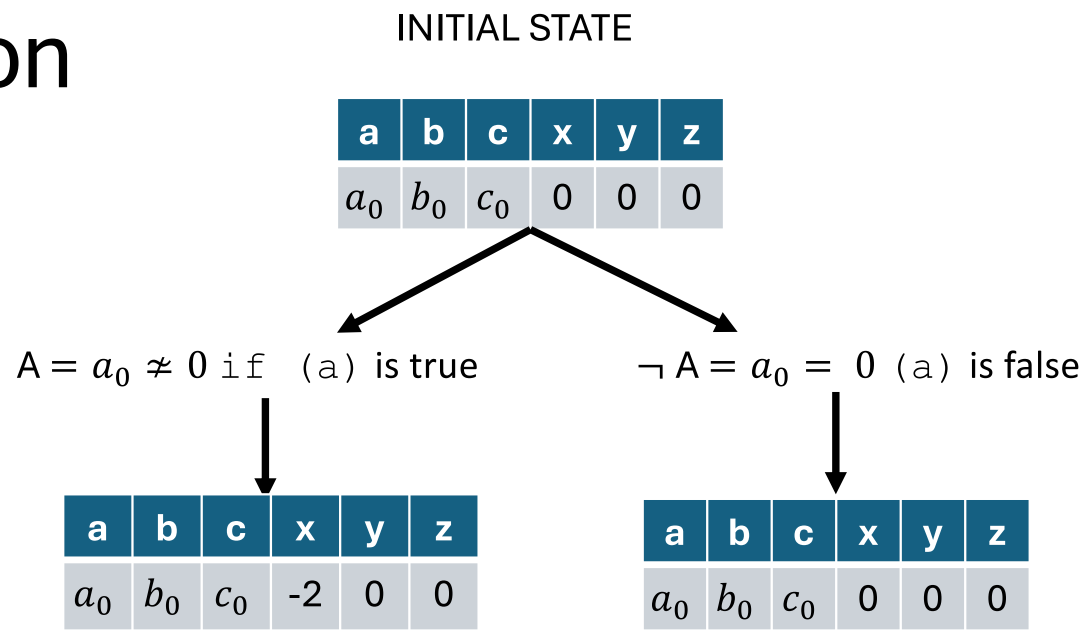
A $= a_0 \neq 0$ `if (a)` is true

A $= a_0 = 0$ `(a)` is false

Simbolic Values a_0, b_o, c_o

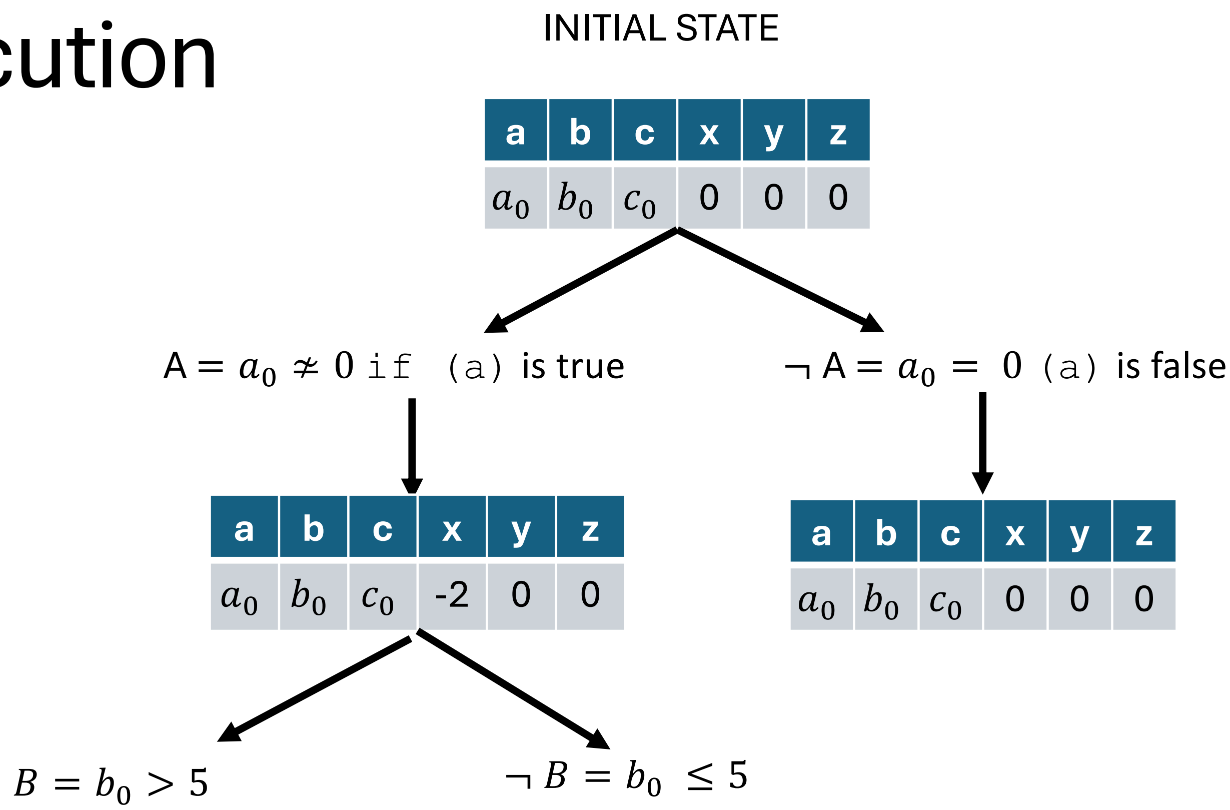
Symbolic execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```



Symbolic execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```



Symbolic execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

INITIAL STATE

a	b	c	x	y	z
a_0	b_0	c_0	0	0	0

A $= a_0 \neq 0$ `if (a)` is true

a	b	c	x	y	z
a_0	b_0	c_0	-2	0	0

$B = b_0 > 5$

$\neg A \wedge (c \neq 0) = \neg A \wedge C$

a	b	c	x	y	z
a_0	b_0	c_0	-2	1	2

$A \vee \neg C$

a	b	c	x	y	z
a_0	b_0	c_0	-2	0	2

$\neg B = b_0 \leq 5$

¬ A $= a_0 = 0$ `(a)` is false

a	b	c	x	y	z
a_0	b_0	c_0	0	0	0

Symbolic execution

FOCUS ON THE RED PATH

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

$$\boldsymbol{A \wedge B \wedge \neg A \wedge C}$$

INITIAL STATE

a	b	c	x	y	z
a_0	b_0	c_0	0	0	0

A $= a_0 \neq 0$ `if (a)` is true

a	b	c	x	y	z
a_0	b_0	c_0	-2	0	0

$\neg$ A $= a_0 = 0$ `(a)` is false

a	b	c	x	y	z
a_0	b_0	c_0	0	0	0

$B = b_0 > 5$

$\neg B = b_0 \leq 5$

$\neg A \wedge (c \neq 0) = \neg A \wedge C$

a	b	c	x	y	z
a_0	b_0	c_0	-2	1	2

$A \vee \neg C$

a	b	c	x	y	z
a_0	b_0	c_0	-2	0	2

Symbolic execution

FOCUS ON THE RED PATH

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

UNFEASIBLE

$\boldsymbol{A \wedge B \wedge \neg A \wedge C}$

INITIAL STATE

a	b	c	x	y	z
a_0	b_0	c_0	0	0	0

A $= a_0 \neq 0$ `if (a)` is true

a	b	c	x	y	z
a_0	b_0	c_0	-2	0	0

$\neg$ A $= a_0 = 0$ `(a)` is false

a	b	c	x	y	z
a_0	b_0	c_0	0	0	0

$B = b_0 > 5$

$\neg B = b_0 \leq 5$

$\neg A \wedge (c \neq 0) = \neg A \wedge C$

a	b	c	x	y	z
a_0	b_0	c_0	-2	1	2

$A \vee \neg C$

a	b	c	x	y	z
a_0	b_0	c_0	-2	0	2

Symbolic execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

INITIAL STATE

a	b	c	x	y	z
a_0	b_0	c_0	0	0	0

A $= a_0 \neq 0$ `if (a)` is true

a	b	c	x	y	z
a_0	b_0	c_0	-2	0	0

$\neg$ A $= a_0 = 0$ `(a)` is false

a	b	c	x	y	z
a_0	b_0	c_0	0	0	0

$B = b_0 > 5$

$\neg B = b_0 \leq 5$

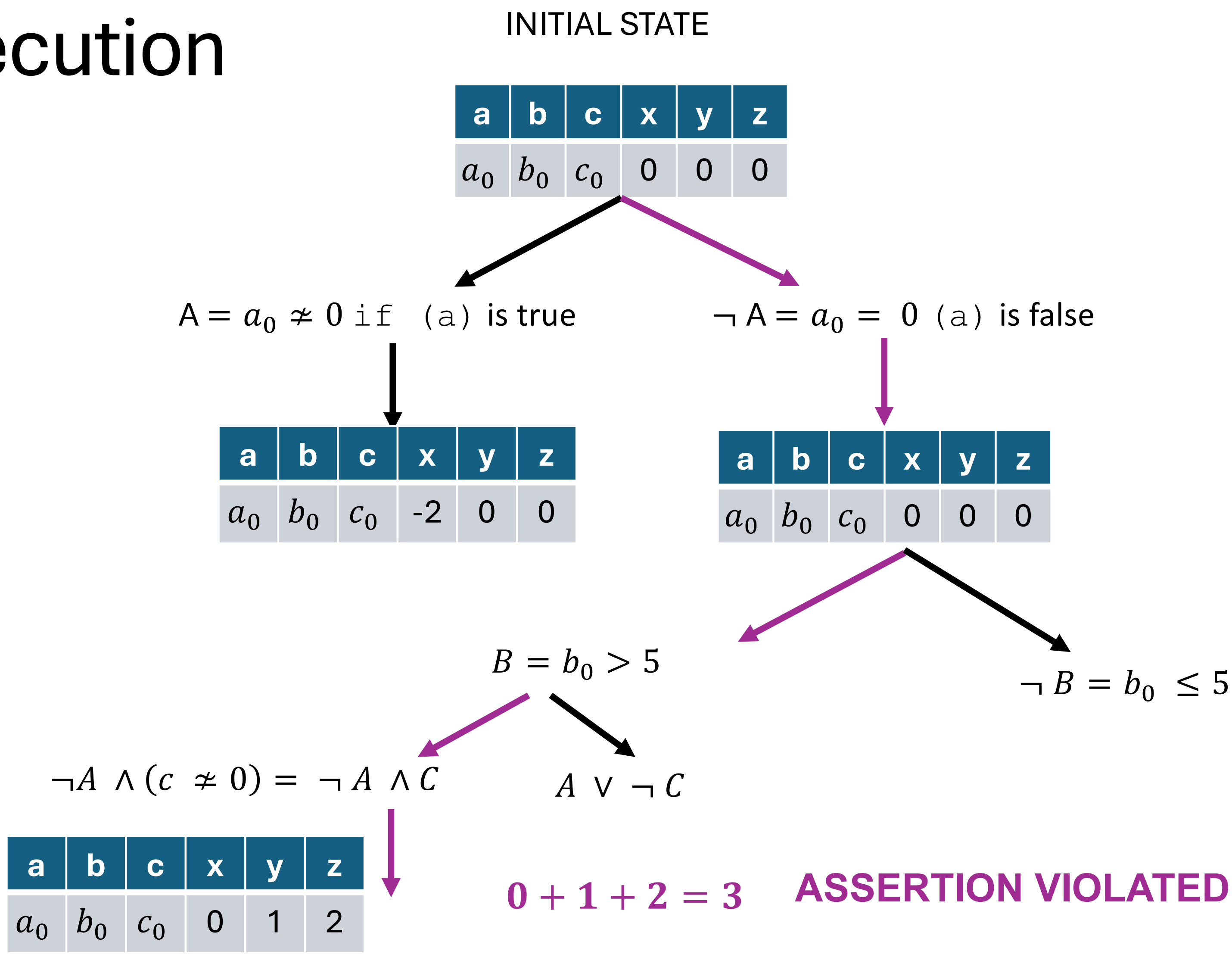
$\neg A \wedge (c \neq 0) = \neg A \wedge C$

$A \vee \neg C$

a	b	c	x	y	z
a_0	b_0	c_0	0	1	2

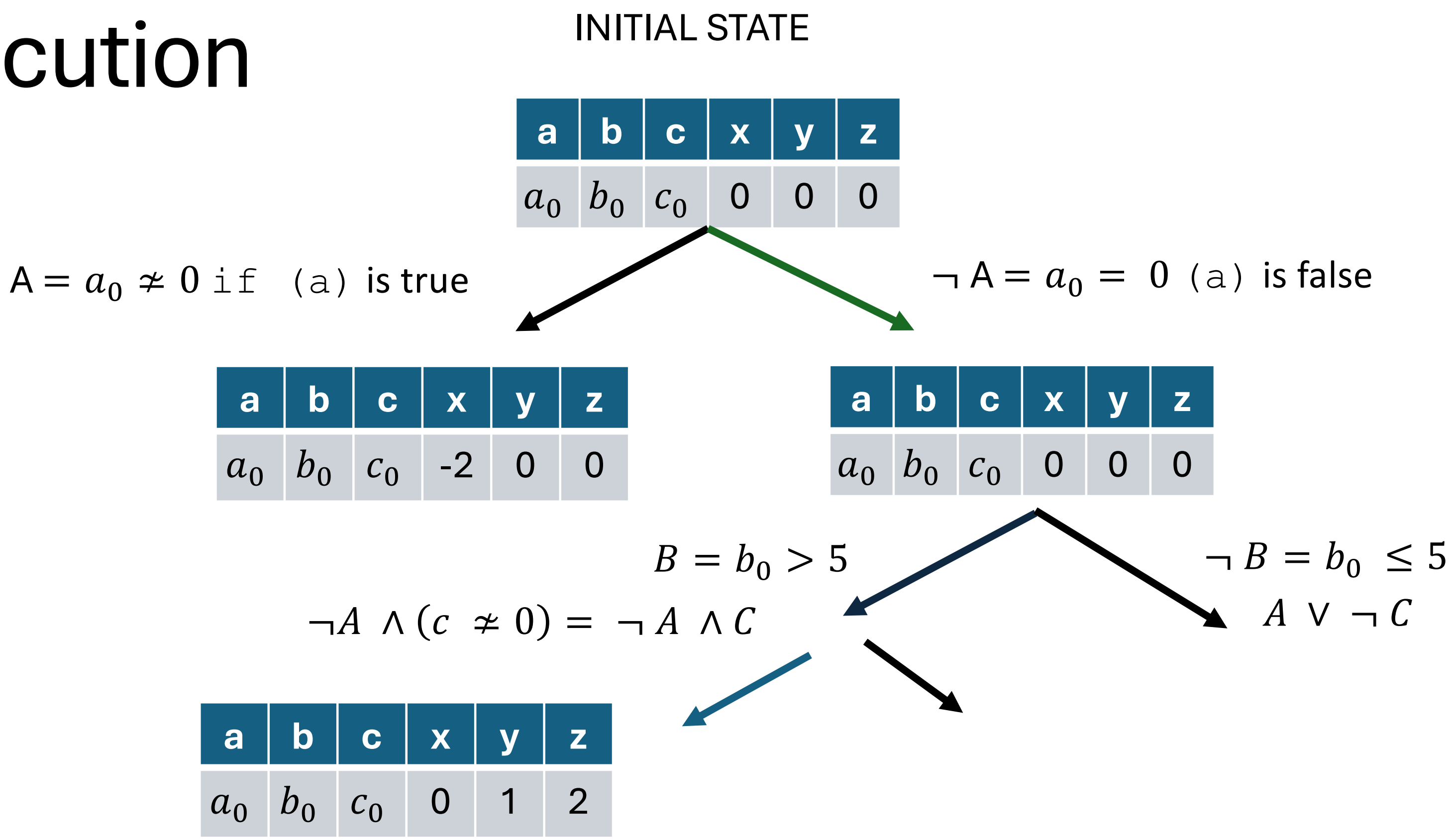
Symbolic execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```



Symbolic execution

```
function f(a, b, c) {
  var x = y = z = 0;
  if (a) {
    x = -2;
  }
  if (b > 5) {
    if (!a && c) {
      y = 1;
    }
    z = 2;
  }
  assert(x + y + z != 3);
}
```

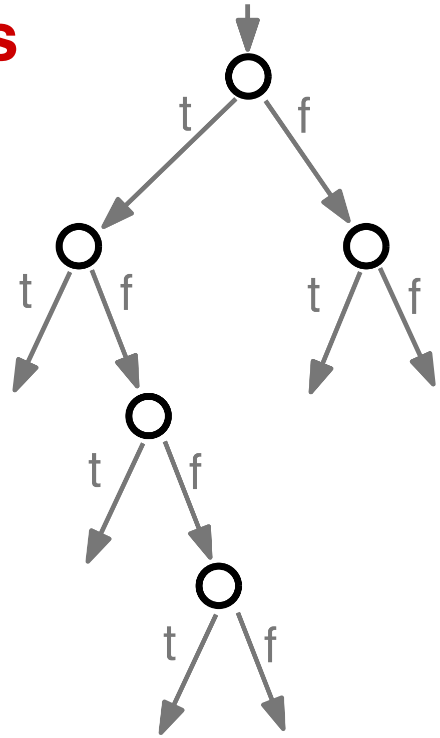


SIMPLER TREE LIKE REPRESENTATION

Execution Trees

All possible execution paths

- Binary tree
- Nodes: Conditional statements
- Edges: Execution of sequence on non-conditional statements
- Each path in the tree represents an equivalence class of inputs



Exercize

Draw the execution tree for this function. How many nodes and edges does it have?

```
function f(x,y) {
  var s = "foo";
  if (x < y) {
    s += "bar";
    console.log(s);
  }
  if (y === 23) {
    console.log(s);
  }
}
```

Symbolic Values and Symbolic States

- Unknown values, e.g., user inputs, are kept symbolically
- Symbolic state maps variables to symbolic values

```
function f(x, y) {
  var z = x + y;
  if (z > 0) {
    ...
  }
}
```

Symbolic Values and Symbolic States

Symbolic Values and Symbolic State

- Unknown values, e.g., user inputs, are kept symbolically
- Symbolic state maps variables to symbolic values

```
function f(x, y) {
  var z = x + y;
  if (z > 0) {
    ...
  }
}
```

Symbolic input values: x_0, y_0

Symbolic state: $z = x_0 + y_0$

SATISFIABILITY OF PATH CONDITIONS

Determine whether a path is feasible:
Check if its path condition is satisfiable

- Done by powerful SMT/SAT solvers
  - SAT = satisfiability,
    SMT = satisfiability modulo theory
  - E.g., Z3, Yices, STP
- For a satisfiable formula, solvers also provide a concrete solution

SATISFIABILITY OF PATH CONDITIONS

Determine whether a path is feasible:
Check if its path condition is satisfiable

- Examples:
  - $a_0 + b_0 > 1$: Satisfiable, one solution: $a_0 = 1, b_0 = 1$
  - $(a_0 + b_0 < 0) \wedge (a_0 - 1 > 5) \wedge (b_0 > 0)$: Unsatisfiable

APPLICATIONS OF SYMBOLIC EXECUTION

- General goal: Reason about behavior of program
- Basic applications
  - Detect infeasible paths
  - Generate test inputs
  - Find bugs and vulnerabilies
- Advanced applications
  - Generating program invariants
  - Prove that two pieces of code are equivalent
  - Debugging
  - Automated program repair

EXAMPLE: Generate Test Inputs

```
function test(x, y) {
 var z = 0;
 if (x > 0) {
  z = z + 1;
 } else {
  z = z - 1;
 }

 if (y == z) {
  assert(false);
 }
}
```

Goal: **symbolically execute** this program to find **inputs (x, y)** that **violate the assertion**.

EXAMPLE: Generate Test Inputs

```
function test(x, y) {
  var z = 0;
  if (x > 0) {
    z = z + 1;
  } else {
    z = z - 1;
  }

  if (y == z) {
    assert(false);
  }
}
```

Goal: **symbolically execute** this program to find **inputs (x, y)** that **violate the assertion**.

Step 1 – Symbolic initialization

At the start:

Variable	Symbolic value
x	x_0
y	y_0
z	0

EXAMPLE: Generate Test Inputs

```
function test(x, y) {
  var z = 0;
  if (x > 0) {
    z = z + 1;
  } else {
    z = z - 1;
  }

  if (y == z) {
    assert(false);
  }
}
```

Goal: **symbolically execute** this program to find **inputs (x, y)** that **violate the assertion**.

Step 2 – First branch: `if (x > 0)`

Two possible paths:

Path 1: $x_0 > 0$

Then $z := 1$

Path 2: $x_0 \leq 0$

Else $z := -1$

EXAMPLE: Generate Test Inputs

```
function test(x, y) {
  var z = 0;
  if (x > 0) {
    z = z + 1;
  } else {
    z = z - 1;
  }

  if (y == z) {
    assert(false);
  }
}
```

Goal: **symbolically execute** this program to find **inputs (x, y)** that **violate the assertion**.

Step 3 – Second branch: `if (y == z)`
Each prior path splits again based on this condition.

Path 1A
Condition: $x_0 > 0 \ \land \ y_0 == 1$
Assertion fails (`assert(false)` triggered).

This is a bug path.

Path 1B
Condition: $x_0 > 0 \ \land \ y_0 \neq 1$
Safe path (no failure).
:

EXAMPLE: Generate Test Inputs

```
function test(x, y) {
 var z = 0;
 if (x > 0) {
  z = z + 1;
 } else {
  z = z - 1;
 }

 if (y == z) {
  assert(false);
 }
}
```

Goal: **symbolically execute** this program to find **inputs (x, y)** that **violate the assertion**.

Path	Condition	Test Input	Assertion
1A	$x_0>0 \wedge y_0=1$	(2, 1)	Fails
1B	$x_0>0 \wedge y_0\neq1$	(3, 0)	Passes
2A	$x_0\leq0 \wedge y_0=-1$	(0, -1)	Fails
2B	$x_0\leq0 \wedge y_0\neq-1$	(-5, 2)	Passes

NEXT LECTURE

- Foundation of (modern) symbolic execution